**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**
**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ**
**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**
**Кафедра информационных систем в искусстве и гуманитарных науках**

**ДОПУСТИТЬ К ЗАЩИТЕ**
**Заведующий Кафедрой информационных систем в искусстве и гуманитарных науках**
**_____(Борисов Н.В.)**
**“ 23 ” мая 2016 г.**

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**
**Основная образовательная программа**
**«Прикладная информатика в области искусств и гуманитарных наук»**
**Направление 230700 «Прикладная информатика»**
**Уровень Бакалавриат**

# «Создание интерактивной игры «Доктор Кто: Тишина в библиотеке»»

**Студента** *Чуркиной Алёны Анатольевны*
_____
*(подпись студента)*

**Руководитель** *доцент СПбГУ, кандидат физ.- мат. наук, с.н.с.*
*Щербаков Павел Петрович*
_____
*(подпись руководителя)*

Санкт-Петербург
2016

# АННОТАЦИЯ

выпускной квалификационной работы
Чуркиной Алёны Анатольевны

название выпускной квалификационной работы
Создание интерактивной игры «Доктор Кто: Тишина в библиотеке»»

Отчет 70 ст., 25 изображение, 12 источников.
РАЗРАБОТКА КОМПЬЮТЕРНОЙ ИГРЫ, СБОРКА СЦЕН, ПРОГРАММИРОВАНИЕ, НАПИСАНИЕ СКРИПТОВ

Целью данной работы является создание интерактивной трехмерной компьютерной игры.

Задачи – сборка сцен, настройка материалов и импортированных объектов, написание необходимых скриптов на языках C# и Javascript.

В процессе работы использовались инструмент для разработки игр Unity 5, текстовый редактор Notepad++.

Автор работы _____ _____
подпись (фамилия, имя, отчество)

Руководитель работы _____ _____
подпись (фамилия, имя, отчество)

# Оглавление

## Определения

*Эскиз* – предварительный набросок, фиксирующий замысел художественного произведения, сооружения, механизма или отдельной его части.

*Локация (игровая локация)* – часть игрового мира, территориально отделенная от других его частей.

*Кат-сцена (Внутриигровое видео)* - это эпизод в компьютерной игре, в котором игрок слабо или вообще никак не может влиять на происходящие события, обычно с прерыванием геймплея.

*Action-adventure с разветвлённым* сюжетом - жанр компьютерных игр, сочетающий элементы квеста и экшена, в котором сюжет зависит от сделанных игроком выборов.

*Скрипт (Script)* - программный код на C# или JavaScript. С его помощью осуществляется управление поведением объектов в сцене.

*Квест* – задание в играх, которое требуется выполнить персонажу (или персонажам) для достижений игровой цели. После его выполнения персонажи получают опыт, деньги, репутацию, вещи, информацию, следующий квест из цепочки и т. д.

*Материал (Material)* – объект в Unity, включающий в себя текстуру и её настройки.

*Компонент (Component)* – функциональная часть любого объекта сцены Unity3D. Определяет поведение объектов в игре. Может быть скриптом или встроенным модификатором Unity3D.

*Инспектор (Inspector)* – меню в Unity3D, которое отображает детальную информацию о текущем выбранном объекте, включая все прикреплённые компоненты и их свойства.

*Пасхалка, Пасхальное яйцо (англ. Easter Egg)* — разновидность секрета, оставляемого в игре, фильме или программном обеспечении создателями.

*Префаб (Prefab)* - тип объектов в Unity, позволяющий хранить весь GameObject со всеми компонентами и значениями свойств. Префаб

выступает в роли шаблона для создания экземпляров хранимого объекта в сцене.

*NPC (Non-Player Character)* - неигровой персонаж, управление которым осуществляется специальной программой.

## Введение

В наши дни компьютерные игры стали настолько популярны, что в США они даже официально признаны отдельным видом искусства с 2011 года.

Первые компьютерные игры появились в 50х-60х годах. Вероятно, в то время люди и не представляли, насколько красочными и удивительными станут игры, и насколько значимое место будут занимать в жизни людей. С тех пор появилось невероятное количество всевозможных жанров, делающих игры абсолютно непохожими друг на друга. Разнообразны и платформы, для которых они разрабатываются. Игры есть и для персональных компьютеров, и для игровых приставок, и даже для телефонов и планшетов.

И обо всём этом нужно помнить, во время создания собственного продукта игровой индустрии. Но как же решить, какой продукт будет больше востребован? Опросить потенциальных игроков и оценить, какие игры пользуются популярностью в мире.

Далее, когда будут определены среда разработки, игровая платформа и некоторыми особенностями создания игровых сцен, можно переходить к следующим этапам. Прежде чем наполнять игру насыщенным сюжетом, сложными персонажами и драматичными диалогами, стоит заняться её основами, некоторым каркасом, на котором всё держится. Что именно будет являться таким каркасом для той или иной игры во многом зависит от её жанра.

Для данного проекта игровые механики включают в себя управление персонажем, его взаимодействие с окружающим миром и интерфейс. Интерфейс в свою очередь включает в себя кодекс знаний игрока, режимы сохранения и загрузки, игровое меню, главное меню. Взаимодействие происходит как с объектами, так и с персонажами (диалоги).

Кроме того необходимо создать игровые сцены, в которых будут происходить события игры.

**Цель выпускной квалификационной работы**

Создание интерактивной компьютерной игры Доктор Кто: Тишина в библиотеке.

**План работы**

1. Выбор среды разработки
2. Исследование целевой аудитории
3. Выбор некоторых алгоритмов генерации элементов игрового мира
4. Сделать выводы на основе полученных результатов и определиться с особенностями создания игры.
5. Импорт моделей и создание игровой сцены
6. Создание игровых механик

# 1. Выбор среды разработки

Первое, с чем стоит определиться для создания своей игры – это, собственно, в какой среде разработки она будет создаваться. Конечно, можно написать игру и с нуля, но в наше время на это мало кто идет.

Сейчас существует целая масса всевозможных игровых движков на любой вкус и цвет, которые позволяют сэкономить силы, время и нервы при разработке. Обычно игровой движок обеспечивает всю основную функциональность, в том числе: «визуализатор» (движок рендеринга), физический движок (обеспечивающий физику игры), звук, графику и многое другое.

Осталось только определиться, который именно из всей массы подойдет конкретно для этого проекта. Стоит учитывать и цену, и возможность найти помощь в сети (всё-таки разработка игр для меня пока что дело новое), и возможности самого движка.

Некоторые варианты были отсеяны из-за своей непопулярности (Torque3D), некоторые – из-за цены (Leadwerks Engine).

В итоге в списке остались лишь 3 гиганта этой индустрии: Unity3D, Unreal Engine 4 и CryEngine 3.

## 1.1 Unity 3D

Unity — это инструмент для разработки двух- и трёхмерных приложений и игр.

«Unity является гибкой и мощной платформой разработки для создания многоплатформенных 3D и 2D-игр и интерактивного контента. Это полная экосистема для всех, кто нацелен на создание бизнеса на разработке высококачественного контента и взаимодействии со своими игроками и пользователями». [1]

*Ключевые особенности:*

- *Цена.* Начиная с 5ой версии у Unity3D появилась полноценная бесплатная версия. Конечно, платную никто не отменял. Но Personal Edition(бесплатная версия) более чем достаточна для инди-разработок.

- *Интерфейс.* Редактор Unity имеет простой Drag&Drop интерфейс, который легко настраивать, состоящий из различных окон, благодаря чему можно производить отладку игры прямо в редакторе. Также интерфейс Unity обеспечивает быстрые итерации. Режим игры в Unity является чрезвычайно мощным инструментом разработки по части быстрого итеративного редактирования во время игры. Можно сразу же оказаться в своей игре: играть и смотреть, как все будет выглядеть в финальной сборке на целевой платформе. Приостановив игру, можно поменять параметры, ресурсы, скрипты и прочие свойства и мгновенно наблюдать результат. Для облегчения отладки можно использовать покадровый просмотр.

- *Мультиплатформенность*. Существует множество платформ, на которых можно развертывать с игровым движком Unity, и их число растет все время. Unity позволяет делать игры для настольных систем и для консолей. Движок Unity обеспечивает поддержку развертывания за один щелчок на платформах PC, Mac и Linux. Теперь независимым разработчикам легче, чем когда-либо ранее, публиковать на консольных платформах. С Unity вы можете бесплатно нацеливаться на PS4, PS3, Xbox One, XBox 360, PlayStation Mobile, PlayStation Vita и Wii U. Ну, а при большом желании созданную игру и вовсе можно запустить на Oculus Rift или Samsung Smart TV. И список платформ, которые поддерживает Unity только продолжает расширяться.

- *Доступ к дополнительным ресурсам.* В Asset Store более чем 10 тыс. уже готовых бесплатных или платных ассетов и инструментов разработки. В том числе и расширения редактора, плагины, окружение, модели и многое другое.

- *Сообщество разработчиков.* Существует множество форумов, где новички могут просить помощи и задавать свои вопросы. И, что самое удивительное, даже получать на них адекватные ответы.

- *Язык написания скриптов.* Скрипты создаются на языках C# и Javascript.

- *Известность.* Один из самых популярных движков. Прославился, в основном, благодаря своему простому и понятному интерфейсу, а не красочным трейлерам.

- *Порог вхождения.* Низкий порог вхождения, благодаря удобному интерфейсу. Вполне возможно создать игру, не написав ни строчки кода. [2]

- *Графика.* В плане графики уступает своим конкурентам.

- *Примеры игр, разработанных на этом движке.* Fallout Shelter, Need for Speed World, Siberia 3.

### 1.2 Unreal Engine 4 (UE4)

Unreal Engine - игровой движок, разрабатываемый и поддерживаемый компанией Epic Games.

Главным образом Unreal Engine разрабатывался для создания FPS (стрелялки от первого лица). Но со временем движок менялся и сейчас уже он пригоден для создания игр любого жанра, главное – умело с ним обращаться. [3]

*Ключевые особенности:*

- *Цена.* 2 марта 2015 года Unreal Engine 4 стал бесплатным. Однако, разработчики игр, как и прежде, должны передавать 5% от прибыли игры компании Epic Games, но при условии, что доходы игры составляют более $3000 за квартал. Безусловно, очень удобно для инди-разработчиков. [4]

- *Интерфейс.* У UE4, в отличие от Unity, есть свой собственный мощный инструмент для дизайна игровых уровней непосредственно в самом движке. Так же UE4 позволяет использовать так называемое визуальное программирование (Blueprints), которое в разы сокращает затраты времени

разработчика. Так, например, управление персонажем можно задать, не написав ни строки кода. Потенциал позволяет написать целую игру, всё ещё не написав ни строчки кода.

- *Мультиплатформенность*. Поддерживает создание игр для Microsoft Windows, Linux, Mac OS и Mac OS X; консолей Xbox, Xbox 360, PlayStation 2, PlayStation 3, PSP, PS Vita, Wii, Dreamcast, GameCube и др., а также на различных портативных устройствах, например, устройствах Apple (iPad, iPhone), управляемых системой iOS и прочих. В этом на сегодняшний день UE4 не уступает Unity.

- *Доступ к дополнительным ресурсам.* Конечно, в интернете ресурсы можно найти для чего угодно. Но, в отличие от Unity, это не встроено ни в официальный сайт UE4, ни, тем более, в сам редактор среды разработки.

- *Сообщество разработчиков.* Благодаря популярности UE, существует множество форумов и сообществ, где можно обсудить данный движок. В том числе и в рунете.

- *Язык написания скриптов.* У Unreal есть свой скриптовый объектно-ориентированный язык программирования, похожий на Java или C++ (UnrealScript). Но можно использовать и C++.

- *Известность.* Unreal достаточно разрекламирован, благодаря своим ярким, красивым и запоминающимся роликам, трейлерам и тизерам. Этот игровой движок всегда на слуху. Известное имя движка так же станет неплохой рекламой и для игры, на нем разрабатываемой.

- *Порог вхождения.* Средний порог вхождения обусловлен тем, что язык написания скриптов специфичный (UnrealScript). Скрипты можно писать и на C++, но порог вхождения в него также довольно высок. Хотя понижает порог вхождение наличие визуального программирования Blueprints. Благодаря нему можно создавать скрипты, технически не написав ни одной строки кода.

- *Графика.* Этот движок обладает высокими графическими возможностями. Их трейлеры и рекламы всегда пестрят яркими спецэффектами и красивыми кадрами.

- *Примеры игр, разработанных на этом движке.* Goat Simulator, BioShock Infinite, Mass Effect 3.

### 1.3 CryENGINE

CryENGINE — игровой движок, разрабатываемый компанией Crytek.

CryENGINE появился с релизом CryEngine (4-ого поколения). Не стоит путать среду разработки CryENGINE с CryEngine 3 SDK (Source Development Kit), по словам разработчиков, с появлением четвертого поколения движка он изменился так сильно, что не может стоять в одной линии со своими предшественниками. [5]

CryENGINE – настоящая машина для создания игр, это профессиональная программа, в которой создаются титанические труды такие как Evolve или RYSE: Son of Rome и т.д.

*Ключевые особенности:*

- *Цена.* Существует бесплатная версия CryENGINE для некоммерческого использования и для обучения. Для коммерческих же надо купить лицензию на договорной основе. Инди-разработчики могут отчислять $9.90 в месяц за каждого пользователя без дополнительного отчисления роялти (как сказали представители VryTek на GDC-2014). [6]

- *Интерфейс.* Этот движок, как и UE4, очень интуитивен и обладает мощными возможностями для дизайна уровней.

- *Мультиплатформенность.* Список уступает по масштабности аналогичному в Unity и UE4. Поддерживаются лишь Windows, PS3, Xbox360, WiiU.

- *Доступ к дополнительным ресурсам.* Так же, как и для UE4, найти дополнительные ресурсы лишь на неофициальной основе.

- *Сообщество разработчиков.* Этот движок ещё достаточно молодой и, соответственно, сообщество ещё не достаточно развито. А это значит, что найти помощь от «старших товарищей» в сети будет проблемно.

- *Язык написания скриптов.* Скрипты создаются на языке C++

- *Известность.* Не так широко известен, как Unity или UE4.

- *Порог вхождения.* Высокий. Чтобы разобраться в нем, придется потратить много времени, это может оказаться достаточно сложным, если вы не имели опыта работы с игровыми движками. [7]

- *Графика.* Графические особенности CryENGINE значительно превосходят возможности первых двух движков, поскольку включают артхаусный свет, реалистичную физику, продвинутую систему анимации и пр.

- *Примеры игр, разработанных на этом движке.* Far Cry, Crysis, ArcheAge.

### 1.4 Вывод

Рассмотрев особенности каждого из этих игровых движков, необходимо остановить свой выбор лишь на одном.

Высокий порог вхождения CryENGINE и малое количество уроков по нему, в итоге, вынуждают от него отказаться.

Выбирая между Unity и Unreal, первым делом хочется сказать «UE4, конечно же! Там ведь такая графика и такие спецэффекты!». Но, подумав немного, понимается, что создаваемый проект вовсе не о том. Снова реклама обманывает и заставляет желать того, что тебе не так уж и нужно.

У меня уже есть некоторый опыт работы и с Unity в целом, и с Javascript в частности. Чего не скажешь ни об Unreal, ни об UnrealScript. Да и техника, имеющаяся в распоряжении, больше подходит для быстродейственной и легкой Unity, чем для тяжеловесного и мощного аппарата Unreal. В игре не

будут задействованы взрывы или красочные спецэффекты, а стилистика была выбрана далекая от реализма. Поэтому преимущества графики оказываются не такими уж и значимыми в данном контексте.

Так что, взвесив все «за» и «против», в качестве среды разработки для создаваемой игры была выбрана программа Unity3D.

# 2. Исследование целевой аудитории

Выбрать, где создать игру – это, конечно, хорошо. Но нужно ещё понять, будет ли кто-нибудь в неё вообще играть. Тематика игры была определена давно и изменениям не подлежала ни в коем случае. В конце концов, название диплома уже утверждено.

На что же ещё смотрят игроки, решая, устанавливать или нет ту или иную игру? Во-первых, это игровая платформа. Ведь если у вас для игр предназначен Xbox, то вы не будете покупать игру под ПК. Во-вторых, жанр игры. По этому поводу у каждого игрока есть свои пристрастия и предпочтения.

Сформулировав вопросы, осталось только найти, кому их задавать. А кому же, как не потенциальным игрокам? В целях данного проекта было разместить соответствующие опросы с пояснениями в игровом сообществе в социальных сетях. Игровое сообщество, в котором был проведен опрос, включает несколько тысяч участников разных возрастов и полов. Полученные результаты будут указаны далее.

## 2.1 Выбор платформы для реализации

Первый вопрос был: «Какие игровые платформы вы предпочитаете?»

В данном случае необходимо было выяснить то, какие платформы пользуются наибольшей популярностью среди наших потенциальных игроков. Так как средой разработки была выбрана Unity3D, то и список платформ включает лишь те, которые можно беспроблемно на ней использовать.

PC - персональный компьютер (имеется в виду любая операционная система: Windows, Mac, Unix или др.).

PS3 - Sony PlayStation 3, игровая приставка Sony седьмого поколения, третья в семействе игровых систем "PlayStation".

PS4 - Sony PlayStation 4, игровая приставка восьмого поколения от компании Sony, четвёртая в семействе стационарных игровых систем "PlayStation".

Xbox One - третья по счёту игровая приставка от компании Microsoft, являющаяся преемником Xbox 360.

Xbox 360 - Xbox 360 - вторая по счёту игровая приставка компании Microsoft, которая последовала за Xbox.

Android\iOS – имеются в виду мобильные телефоны и планшеты.

Wii U - название новой игровой консоли Nintendo, которая является преемником Wii.

В опросе (рис. 1) приняли участие 148 человек. Из них 110 проголосовали за ПК, 27 за консоли и 11за мобильные устройства.


Рисунок 1 – Опрос аудитории, выбор платформы

Победа ПК в сердцах фанатов над остальными платформами видится мне неоспоримой. 74,3% довольно внушительная цифра, так что было принято решение изначально делать игру для персональных компьютеров. Конечно, мультиплатформенность Unity позволит в будущем перенести полученный продукт и на другие платформы, но не всё сразу. С чего-то нужно начинать.

Не стоит забывать и о тех 18,2%, которые проголосовали за различного рода приставки. Человеку, давно и сильно привыкшему играть, используя джойстик, будет довольно сложно перестроиться и вообще начать играть в игру, где доступны лишь мышь и клавиатура. Именно поэтому, для проекта был выбран ещё и альтернативный способ управления персонажем и игрой в целом. Игрок будет иметь возможность сменить способ управления через меню.

## 2.2 Определение особенностей дерева сюжета

Вторым поставленным вопросом к аудитории было: «Какой игровой жанр (из указанных) вы предпочитаете?»

Очевидно, что среди вариантов ответа не абсолютно все возможные жанры, а лишь те, которые можно реализовать в пределах выбранной тематики.

Пожалуй, стоит, наконец, добавить и пару слов о самой концепции проекта. Компьютерная игра создается по мотивам британского телевизионного сериала «Доктор Кто» (эпизод «Тишина в библиотеке»). Главный герой (Доктор Кто) – путешественник во времени и пространстве, отправляется во всевозможные приключения, где спасает людей, инопланетян и прочих существ. В эпизоде, который адаптируется для игры, действие происходит в огромной библиотеке. Основные персонажи эпизода были оставлены соответствующими оригиналу, а вот сценарий планировалось изменять в соответствии с выбранным жанром. Доктор Кто – это в основном головоломки, быстрые решения и много беготни. В таком случае очевидно, что ни стрелялкой (FPS), ни стратегией быть не может.

По изначальной задумке одна из целей игры – дать возможность фанатам на какое-то время и самим стать Доктором, спасителем и героем. А это значит, что никаких мультиплееров – ведь не может в одном и том же месте, спасая одних и тех же людей вдруг оказаться десяток совершенно одинаковых героев.

И так, после анализа и отсеивания были выбраны следующие жанры:

1) Квест (или приключенческая игра) - один из основных жанров компьютерных игр, представляющий собой интерактивную историю с главным героем, управляемым игроком.

   В этом случае игрок получает много головоломок, много разговоров и ещё больше загадок. Но, обычно, подобные игры лишены возможности проверить свою скорость реакции, да и как-то повлиять на сюжет, кроме «прошел»/«не прошел» не представляется возможным.

   Примеры: Syberia, Syberia II.

2) Аркада - компьютерные игры с нарочно примитивным игровым процессом. Целью в них является сбор всех бонусов или набор максимально возможного количества очков.

   Скорость реакции – да. Беготня – да. Головоломки - вряд ли.

   Обычно игры этого жанра являются скорее «убивалкой времени», то есть чем-то, во что люди играют просто когда им скучно, не особо вдаваясь в сюжет и детали. Да и каждый уровень похож на следующий, лишь становясь незначительно сложнее раз за разом.

   Примеры: Super Mario Bros, Sonic Heroes.

3) Action-adventure - жанр компьютерных игр, сочетающий элементы квеста и экшена.

   В этом жанре можно задействовать и быстрые решения, и беготню, и головоломки.

   Казалось бы, чего ещё нужно? Но всё не то. Сюжет давно написан и для всех един: либо вы выживаете и прошли игру, либо умираете и пробуете снова.

   Примеры: Assassin’s Creed III, Tomb Raider.

4) Action-adventure с разветвленным сюжетом - жанр компьютерных игр, сочетающий элементы квеста и экшена, в котором сюжет зависит от сделанных игроком выборов.

Последнее время всё популярнее становятся игры с несколькими концовками, где даже малейшее твое решение может привести к изменениям (большим или маленьким, но всё же).

Примеры: The Wolf Among Us, Heavy Rain

5) Ни один из перечисленных - если ни один из указанных вариантов вам не интересен.

Ведь в мире полно и других жанров. Стоит проверить, не обречена ли игра на провал с самого начала.

В опросе приняли участие 130 человек (рис. 2). Из них 51,5% (67 человек) проголосовали за разветвленный сюжет, 17,7% (23 человека) за Action-adventure без разветвленного сюжета, 15,4% (20 человек) за квест, 10% (13 человек) против всех вариантов и лишь 5,4% (7 человек) за аркады.


Рисунок 2 – Опрос аудитории, выбор жанра

В общем-то, ожидания оправдались, и аудитория действительно предпочитает Action-adventure разветвленным сюжетом. Это неудивительно, ведь каждому хочется верить, что все эти диалоги не просто так и влияют хоть на что-то.

Так что, было принято решение и наш игровой проект реализовывать в рамках этого жанра.

# 3. Выбор некоторых алгоритмов генерации элементов игрового мира на примере алгоритмов построения библиотеки

Кроме того, чтобы определить среду разработки, платформу и жанр, нужно было также решить и то, какими алгоритмами создавать саму игровую среду.

Основные действия игры происходят в библиотеке, поэтому на её примере будет проще всего разобрать этот вопрос.

С самими стеллажами никакой проблемы с выбором вариантов нет. В библиотеках, обычно, все полки одинаковые, ведь изготавливают их массово, по одному дизайну. Поэтому, для них вполне разумно использовать одну и ту же продублированную модель. В локациях не предусматривается лабиринтов, поэтому расставить их вручную так же не составит большой проблемы.

Но как же сами книги на полках. Не расставлять же и их все вручную. Из-за неоптимальности и слишком высоких затрат ресурсов этот вариант исключается сразу. Стоит выбрать что-то получше.

Первый и самый очевидный вариант: просто продублировать один красивый стеллаж. Как оказалось, не самый приятный глазу прием.

Для примера привожу скриншоты из игры Dragon Age: Inquisition (рис. 3). Три совершенно одинаковых стеллажа стоят рядом. Каждый из них,


Рисунок 3 – Книжные стеллажи в игре Dragon Age: Inqusition

неоспоримо, хорош и аккуратно сделан. Но вместе это выглядит нелепо. Наверное, эти стеллажи мелькают в игре только один раз, и на них даже никто не обращает внимания, возможно, скажете вы. Но нет. Точно такие же стеллажи в другой комнате, в другом освещении, но с совершенно такими же

Рисунок 4- Повторяющиеся стеллажи в игре Dragon Age: Inqusition

полками, заставленными абсолютно такими же книгами (рис. 4).

Можно представить и ужаснуться, как однообразно будет выглядеть целая библиотека, заставленная идентичными шкафами. Хотя. Зачем представлять? Пример всё из той же игры (рис. 5).

Рисунок 5 – Библиотека в игре Dragon Age: Inqusition

Скажем честно, им простительно. Всё же, это игра о сражениях с демонами и драконах, а не о библиотеках. Но, совершенно очевидно, что для создаваемой игры такой вариант совершенно неприемлем.

Другой вариант: автоматическая генерация книг. В том числе случайное изменение некоторого их размера (высота, толщина).

Скриншоты из игры The Elder Scrolls: Skyrim (рис. 6). Очевидно, что книги на полках расставлены в случайном порядке и книжные полки не повторяются идеально в том же порядке.

Рисунок 6 – Книжные стеллажи в игре The Elder Scrolls: Skyrim

На основе этих фактов, было принято решение, что для своей реализации библиотеки будут использована случайная генерация книг. Это должно дать библиотеке возможность выглядеть живее, правдоподобнее и интереснее для игрока.

# 4. Импорт моделей и создание игровой сцены

Игровые сцены – очень важный элемент любой игры. В некоторых из них, игроки готовы бегать долго, изучая детали и любуясь видами, иногда даже забывая на время о цепочке квестов. Другие же просто радуют глаз и оказываются приятным фоном.

У Unity есть много удобных и простых в использовании средств для создания, изменения и настройки игровых сцен на свой вкус. Например, `terrain` для создания ландшафтов, `skybox` для создания неба, встроенная в Unity динамическая поверхность воды.

К сожалению или к счастью, основные действия игры происходят в помещениях (квартира, библиотека). Поэтому придётся отложить и terrain, и динамическую воду до лучших времен и других проектов.

При большом желании, каждую сцену можно создавать сразу в 3ds max и импортировать её целиком, не разделяя на объекты. Но, в таком случае, может возникнуть ряд трудностей при настройке материалов, обращении к объектам через скрипты и в принципе настройкой и небольшими изменениями сцены.

Поэтому было принято решение импортировать модели окружения по отдельности и собрать полноценную сцену уже в Unity.

## 4.1 Импорт моделей

В вопросе импорта Unity очень удобна и проста в использовании.

Во-первых, импортировать можно несколькими способами (каждый сам выбирает, какой из них удобней лично для него). Можно просто положить соответствующие файлы в папку проекта (через Проводник, Total Commander и т.д.), можно перетащить файлы из Проводника в окно Project интерфейса Unity или же выбрать в меню пункт Assets>Import New Asset.

Импорт в Unity понимает такие экспортируемые форматы как .FBX, .dae (Collada), .3DS, .dxf и .obj. Также поддерживается и импорт напрямую из некоторых популярных 3D редакторов (3ds max, Maya, Blender и т.д.).

У обоих этих способов есть свои плюсы и минусы.

С одной стороны, конечно, удобно без особых проблем и конвертаций сохранять сразу файлы формата .max в проект Unity. Этот способ простой и очевидный. Но для этого на компьютере должны стоять и Unity, и используемый 3D редактор (чем я, к сожалению, не располагаю). Также, в этом случае, в файле хранится вся возможная и доступная информация о модели. В то время как при экспорте из редактора через другие форматы можно настроить, что именно должно быть сохранено и перенесено. Например, не переносить свет, анимацию, камеры или перенести только один объект из сцены, а не всю сцену целиком. [8]

По этим причинам было решено импортировать объекты в формате .FBX.

Стоит также отметить, что импорт других объектов (изображений, аудио- и видеозаписей различных форматов) также предельно прост и аналогичен описанному ранее.

Все игровые модели были созданы и текстурированы Екатериной Яценко, после чего экспортированы в формате .FBX и добавлены в проект.

Импорт самих моделей прошел успешно и без нарушений (оси сохранились, формы не деформировались). Но текстурам повезло не так сильно.

При переносе модели Unity генерирует материалы на основе тех, которые были у объекта в 3D редакторе. Но не всегда это у неё получается удачно.

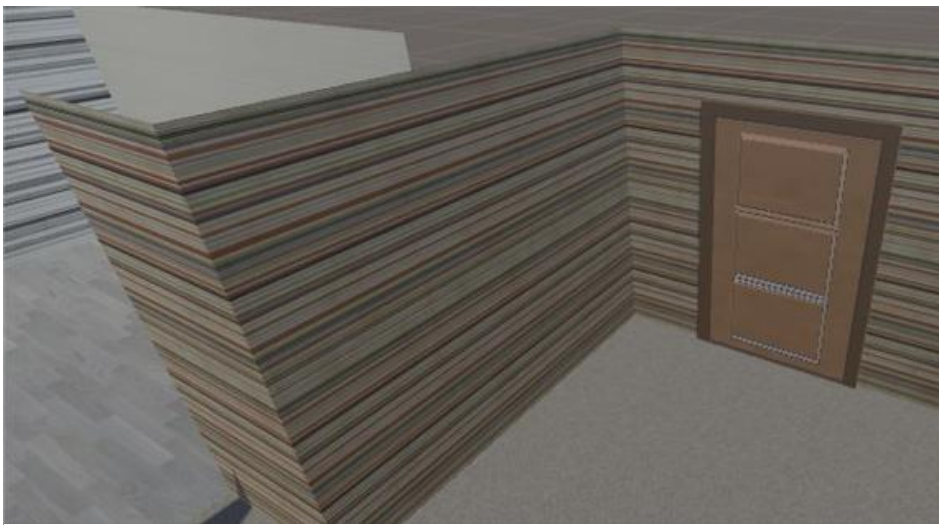

Рисунок 7 – Сбой текстуры при переносе в Unity

Например, при импорте стен комнаты, материал не стал переворачивать текстуру (как это было сделано в 3ds Max) и это пришлось настраивать вручную (рис. 7). Также, с некоторыми моделями нужно было настраивать масштабирование текстуры, характеристики, отвечающие за блеск и отражения.

Кроме того, для текстур `NormalMap` необходимо было дополнительно указать, что они являются именно картами нормалей, а не обычными текстурами. Иначе они не осуществляли бы свои функции. Но надо сказать спасибо Unity за то, что об этом нюансе она сообщила и даже указала, как это исправить (рис. 8).

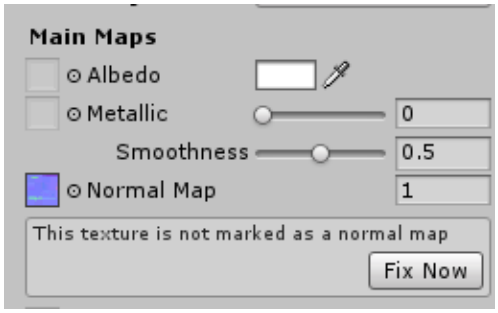


Рисунок 8 – Подсказка Unity по поводу карты нормалей

Для импорта объектов в Unity3D важной и полезной оказывается вкладка Import Settings, показанная в инспекторе. Хотя изначальных настроек по умолчанию вполне достаточно, среди них есть и несколько особенно интересных и оказавшихся полезными:

1. Scale - это фактор масштаба, используемый для компенсации разницы в единицах измерения между Unity и инструментом 3d моделирования. Он меняет масштаб всего файла. Физический движок Unity масштабирован так, что 1 единица измерения эквивалентна 1 метру. Это особенно важно, когда необходимо получить корректное физическое поведение.
2. Generate colliders - эта опция создает сетку коллизии, что позволяет модели сталкиваться с другими объектами. [9]

Кроме самих моделей и текстур, в проект также надо было перенести и анимацию некоторых объектов (полицейская будка для главного меню).

Вкладка Input Settings также содержит вкладку Animations, которая оказывается полезной именно в такой момент.

В этой вкладке можно просмотреть существующую у модели анимацию, при необходимости обрезать её или разделить на несколько частей, каждой из которых присвоив свое имя.

В случае с полицейской будкой в этом не было необходимости и вполне достаточным было лишь убедиться, что анимация перенесена, и указать ей подходящее имя (policeBoxAnimation).

### 4.2 Создание игровых сцен

Когда все модели были перенесены и корректно настроены, сцены оставалось только собрать. Так как действия в создаваемых сценах происходят только внутри комнаты, и игрок не выходит за пределы стен, было решено не создавать `terrain`, а просто поместить в сцене `skybox`, который через окна помещения будет виден, но не будет слишком сильно отвлекать.

Также необходимо было разместить свет. Был использован `Point light`, как наиболее похожий на комнатные лампы и люстры. Этому типу освещения можно задать направление лучей, радиус освещаемого участка, интенсивность света и цвет. Кроме того использовался `Direct Light` для создания общей атмосферы помещений.

Для каждой сцен свет настраивался индивидуально с учетом того, что будет происходить в той или иной комнате, на какие её части игрок должен обратить внимания, и какое общее впечатление должно у него остаться от помещения.

Например, в комнате с диваном одна люстра, поэтому у неё большой радиус действия и высокая интенсивность света. В то время как две небольшие люстры на кухне обладают небольшим радиусом действия и более низкой интенсивностью. Были добавлены аналогичные источники света на кухне под полками, чтобы обеспечить более интересное освещение находящимся

там объектам (рис. 10). Ещё один источник света был добавлен для экрана телевизора. Он обладает ещё меньшей интенсивностью, чем остальные упомянутые, но широким радиусом. Так же его лучи имеют голубоватый, а не белый цвет. В начальном режиме он выключен и включается только тогда, когда пользователь активирует телевизор.

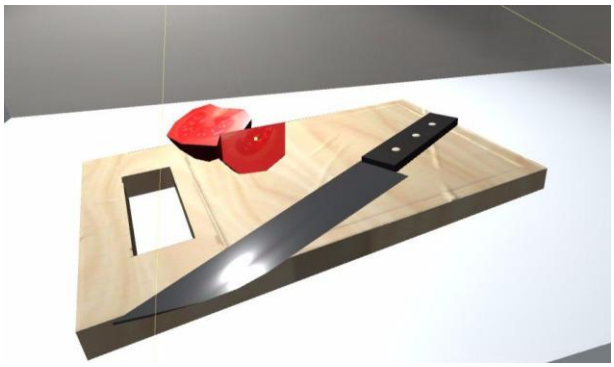

Рисунок 10 – Объект, освещённый дополнительными лампами

В итоге на основе эскиза из 3ds max и концепт-артов помещения была создана сцена (Рисунок 11).

Рисунок 11 – Созданная игровая сцена квартиры

В то время как квартира выглядит теплой и спокойной (в том числе из-за освещения и из-за особенностей использованных материалов), комнаты библиотеки необходимо было сделать подчеркнуто мрачными и

таинственными (рис. 12). В комнатах библиотеки источники света холодного оттенка вместо теплого. Материалы объектов настроены таким образом,

Рисунок 12 – Созданные игровые сцены библиотеки

чтобы их тени имели такой же холодный, таинственный оттенок, нагнетая обстановку.

Кроме того в некоторых случаях оказалось необходимым использовать инструмент создания тумана `Fog`. Создание тумана черного цвета позволило создать ощущение сгущающейся темноты и длинных темных коридоров (хотя на самом деле их длина довольно ограничена и не так уж велика).

### 4.3 Настройка камеры

Серьезная проблема игровой сцены в частности и игры в целом – это стилизация окружения. С одной стороны, она не должна быть слишком реалистичной (если она всё же заявляется как стилизация). Но с другой стороны слишком пластмассовую и фальшивую картинку игроку наблюдать будет тоже неприятно.

Казалось бы, этим должны озадачиваться исключительно дизайнеры и 3D-моделеры. Но нет, и Unity-разработчик может приложить к этому свою руку.

Не важно, насколько реалистичным будет создан тот или иной объект окружения. Без постэффектов ощущения погружения не получится. Это нам доказывают и 3D-мультфильмы, которые выглядят так удивительно и сказочно с постобработкой и эффектами, и так несуразно без неё. В Unity есть целый ряд средств для улучшения «картинки».

Всё, что видит игрок на экране, на самом деле он видит через камеру, которая следует за персонажем. В игровой сцене Unity эта камера является объектом и, соответственно, её можно дополнить различными компонентами.

На самом деле, всевозможных эффектов постобработки, используемых для различных целей очень много. Есть эффект, полностью размывающий экран, есть эффект motion blur, который размывает лишь объекты при движении. Какие-то из них более ресурсоемкие, какие-то менее. А некоторые эффекты,такие как расширенный динамический диапазон, доступны лишь для Pro версии Unity.

Рисунок 13 - Игровая сцена без эффектов

Рисунок 14 – Игровая сцена с эффектами

Попробовав некоторые из них, были выбраны наиболее подходящие для создаваемого проекта:

1. Глубина резкости (Depth of field) - эффект выборочно размывает области изображения, находящиеся вне фокуса. [10]

2. SSAO (Screen Space Ambient Occlusion) - добавляет мягкие тени к трещинам и объектам, которые находятся в непосредственной близости друг от друга. [10]

На скриншоте видно как выглядит изображение без эффектов (рис. 13) и с эффектами (рис. 14).

Таким образом, были ипортированы необходимые объекты из 3ds max, собрана и настроена сцена. Созданы и настроены материалы для каждого обхекта, настроена их необходимая анимация. Для каждой сцены был тщательно настроены и продуманы источники света в соответствии с сюжетом и событиями игры. Для комнат, где это было необходимо, был настроен туман. Также была настроена камера для получения удовлетворяющего всем требованиям результата.

# 5. Создание игровых механик

Закончив с созданием игровых сцен, в них уже можно добавить игрока, чтобы настроить его возможности. Для начала в игре использовался персонаж от первого лица исключительно потому, что это позволяет легче осуществлять тестирование и проверку окружения на ошибки. Кроме того, дает независимость от того, закончен ли уже персонаж 3D-моделером или нет. Для персонажа основными игровыми механиками в разрабатываемой игре является управление (с клавиатуры и контроллера) и взаимодействие с игровыми объектами. Кроме того необходимо обеспечить возможность пополнять и просматривать кодекс. Персонаж так же должен иметь возможность вести диалоги с NPC, учитывая их возможную разветвленность. Для полноценной игры необходимы такие базовые вещи, как главное меню, игровое меню, загрузка и сохранение. И не стоит забывать и об уже принятом решении генерировать книги на полках библиотеки программным образом.

## 5.1 Управление персонажем

Первым делом, чтобы проверять хоть какие-то функции игры, нужно, чтобы персонаж умел перемещаться в пространстве.

Благо, в Unity есть стандартные контроллеры персонажей (от первого и от третьего лица), которые вполне можно использовать. Они не перегружены лишним функционалом, но вполне выполняют возложенную на них функцию. Единственное, что нужно было добавить, это возможность управлять персонажем при помощи контроллера и настроить анимацию персонажа. В Unity есть возможность добавлять новые назначения кнопок через `Edit>Project Settings>Input`. Через эту вкладку можно добавить, настроить и отредактировать назначения кнопок. Также есть возможность указать альтернативные кнопки для одного и того же действия. [11] Таким образом, в коде скрипта указывается загадочное Activation, а в настройках Input уже прописано, какие именно кнопки этому Activation

соответствуют (в том числе, кнопка на клавиатуре и альтернативная ей на контроллере).

Другая проблема управления персонажем – объекты. По умолчанию, у импортированных моделей нет сетки коллизий, так что игрок просто проходит их насквозь, не столкнувшись.

Во вкладке Import Settings есть опция `Generate colliders`. Она-то и создает сетку коллизии для импортированной модели. [12]

### 5.2 Взаимодействие с объектами

После того как герой научился перемещаться в пространстве и не проходить сквозь объекты, необходимо было научить его с этими объектами взаимодействовать.

В игре присутствуют объекты двух типов:

- информационные объекты
- объекты-экраны

*Информационные объекты.* Этот тип объектов позволяет игроку получить некоторую информацию. Часть этой информации довольно бесполезна и представляет из себя лишь шутки, пасхалки или предупреждения (например, что дверь не активна и выйти через неё не удастся). Другая же часть будет давать игроку возможность изменить ход сюжета (например, дополнительная реплика во время диалога).

*Объекты-экраны.* Эти объекты представляют из себя экраны телевизоров, терминалов или компьютеров. При взаимодействии игрока с ними, экран активируется и на нём отображается заранее подготовленная видеозапись.

Для универсальности и удобства для объектов обоих типов используется одни и те же скрипты. Отличаются объекты друг от друга только используемым тегом. Тег «Tv» для объектов-экранов и тег «InfoThings» для информационных объектов.

**Скрипт:** GamingMenu, UsableThing, XMLtest, testSaveLoad.

**Использовался язык программирования:** C#.

**Объект:** Скрипт UsableThing лежит на всех объектах, доступных для взаимодействия, скрипты GamingMenu, XMLtest и testSaveLoad лежат на игровом персонаже.

**В игре:** Когда персонаж оказывается лицом к объекту, с которым возможно взаимодействие и при этом он достаточно близко для взаимодействия, на экране отображается темная панель внизу экрана. На этой панели выводится текст-подсказка. Для каждого объекта текст-подсказка индивидуальный.

Если это информационный объект, то в подсказке, соответственно, будет указана форма взаимодействия с ним.

Например, холодильник и книжный шкаф в одной из созданных сцен (рис. 14). Для книжного шкафа указана подсказка «Прочитать», в то время как для холодильника указывается подсказка «Осмотреть».


Рисунок 14 – Пример взаимодействия с информационным объектом

После того, как объект будет активирован (игрок нажмет кнопку E чтобы «осмотреть» или «прочитать»), на экране появится окно с соответствующей


Рисунок 15 – Пример всплывающих окон с информацией

информацией из кодекса (получается через скрипт XMLtest) (рис. 15). Через скрипт testSaveLoad программа отмечает в текущей игре, что прочитанная запись была добавлена. Так же в углу экрана отображается сообщение о том, что запись была добавлена в кодекс.

Если же это объект-экран, то обычно подсказка указывает, какую кнопку нужно нажать, чтобы его активировать (рис. 16).


Рисунок 16 – Пример взаимодействия с выключенным объектом-экраном

После активации текст подсказки указывает на то, как этот объект деактивировать (Рисунок 17).

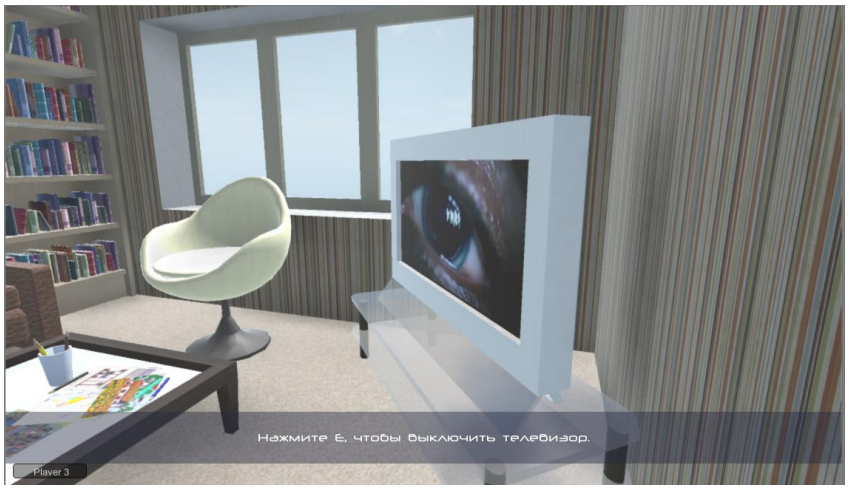


Рисунок 17 - Пример взаимодействия с включенным объектом-экраном

Для анимации экрана телевизора была использована `MovieTexture`, позволяющая использовать видеозапись как текстуру.

Аналогично с информационными объектами, запись добавляется в кодекс через скрипт testSaveLoad, а в углу экрана отображается, что запись была успешно добавлена.

Благодаря созданным основам управления игрой и персонажем, стало возможно тестировать игровой мир и развивать его дальше.

### 5.3 Генерация книг

Во время создания скрипта, обеспечивающего автоматическое генерирование книг на полках, необходимо было учитывать множество тонкостей и нюансов.

В частности, необходимо было преодолеть сложности расположения книг, когда стеллаж не в своем изначальном положении, а, например, повернуть в плоскости XY. Кроме того необходимо было учесть особенности расположения наклонных книг. Изначально была лишь один этап генерации случайного числа (если 1 – наклонная книга, если 0 – обычная). Но в таком случае оказывалось, что примерно половина книг на полке оказывалась наклонной. И это происходило почти на каждой полке. Выглядело неправдоподобно. Поэтому, во-первых была добавлена проверка на то, чтобы две наклонные книги не могли стоять рядом, во-вторых, их количество для каждой полки было ограничено случайным числом от нуля до трех. Но в таком случае были полки, где наклонных книг не было вообще и полки, где наклонные книги были, но все они находились только в начале полки. Поэтому пришлось добавить ещё один этап генерации случайных чисел – номер книги, после которой могут появиться наклонные книги на полке. Для каждой полки он генерируется индивидуально.

Оказалось, что если генерировать только высоту и ширину, общая книжная масса выглядит недостаточно живо и правдоподобно. Поэтому был так же добавлен ещё один параметр – отступ от края полки.

**Скрипт:** testBooks.

**Использовался язык программирования:** Javascript.

**Объект:** Скрипт testBooks лежит на игровом персонаже.

**В игре:** При настройке скрипта, ему подаются префабы книг (стоящая ровно и стоящая под наклоном) и допустимые варианты обложек. Программа находит все полки (по тегу) и автоматически выставляет книги от начала полки до её конца. Высота, ширина книги и отступ от края генерируются в обозначенных в программе диапазонах. Обложка случайно выбирается из допустимых вариантов (рис. 18).

Перед началом расстановки книг на полке программа генерирует случайное количество наклонных книг на полке в диапазоне от нуля до трех включительно. Если количество наклонных книг ненулевое, то программа так же генерирует некоторые дополнительные параметры расположения

Рисунок 18 – Книжные полки, сгенерированные программно

наклонных книг на полках. Программа учитывает, чтобы две наклонные книги не стояли рядом, чтобы книги не оказывались каждый раз в одних и тех же участках полки и т.д.

### 5.4 Создание главного меню

Меню должно быть удобным, стильным и соответствовать тематике игры. Первое, что человек видит, запуская игру – это главное меню. Порой, если оно не впечатляет, то и в игру уже играет он не так восторженно, и впечатление не то. Ведь главное меню – это та одежка, по которой игру встречают.

Главный герой создаваемой игры – путешественник во времени и пространстве, который перемещается по миру в синей полицейской будке.
В ходе данной квалификационной работы создавалась лишь первая глава игры по мотивам одного из эпизодов британского сериала Доктор Кто. Но после выпуска из университета планируется продолжить её разработку и создать ещё несколько глав, в том числе и по мотивам других эпизодов всё того же сериала.
Именно поэтому было принято решение не привязывать дизайн интерфейса к конкретной создаваемой части игры, а сделать его универсальным, подходящим для всего сериала в целом.
В ходе данной работы были выполнены лишь действия по реализации продуманного дизайна меню, сам же дизайн был продуман и составлен Екатериной Яценко. Дизайн меню (и главного, и игрового) был сделан футуристичным в космических цветах.
Задумавшись о главном меню, невольно вспоминаешь, а какие же они в других играх? В большинстве популярных игр сейчас главное меню строится по принципу: красивый анимированный зацикленный фон и кнопки в виде текста на полупрозрачном фоне (Рисунок 19).


Рисунок 19 – Примеры главного меню в современных играх.
В Dragon Age: Inqusition и Fallout4 соответственно

Будка, в которой путешествует главный герой, уже давно стала символом сериала Доктор Кто (по мотивам которого и создается игра). Именно поэтому, когда появился вопрос «А что же поставить этим анимированным фоном?», долгих размышлений не последовало. Определившись с концепцией главного меню, можно было переходить к его созданию.

### 5.4.1 Главное меню

С новой версией Unity (Unity 5) создание меню стало намного проще. Некоторые вещи, которые при старой версии необходимо было писать вручную, теперь легко осуществляются за счет интерфейса.

**Скрипт:** MainMenu.

**Использовался язык программирования:** C#.

**Объект:** Скрипты MainMenu лежит на холсте, в пределах которого размещаются все составляющие главного меню.

**В игре:** Не зависимо от размеров окна игры, меню всегда отображается в левом нижнем углу, у неактивных кнопок шрифт меняет цвет на темно-синий (Рисунок 11).

При нажатии на кнопку «Новая игра» появляется окно ввода имени (название сохранения). Текущей игрой становится новый слот игры. Изначально все записи кодекса считаются недоступными, а уровень доверия с каждым персонажем равен нулю, указываются начальные уровень и координаты по умолчанию.

При нажатии на кнопку «Загрузка» вместо главного меню появляется полупрозрачное меню имеющихся сохранений. Список формируется из внешнего файла, в котором хранятся сохранения. Чтобы загрузить необходимое сохранение нужно нажать на него (изображение над списком изменится), после чего нажать на кнопку «Загрузить» в правом нижнем углу.

При нажатии на кнопку «Загрузить» появляется окно подтверждения с кнопками «Ок» и «Отмена». При отмене игрок возвращается в основное меню. Иначе данные о текущей игре заменяются на выбранные, и загружается игровой уровень, указанный в сохранении. Если необходимо вернуться в главное меню, в правом нижнем углу есть кнопка «Назад».

При нажатии на кнопку «Выход» происходит выход из игры.

### 5.4.2 Создание анимированного фона главного меню

Зацикленная анимация для полицейской будки была создана Екатериной Яценко в 3ds max и перенесена в Unity (см. выше).

Так как будка должна повторять одну и ту же анимацию с момента, когда сцена загружается и до момента, когда производится переход на другую сцену или выход из игры, смысла писать скрипт, управляющий её анимацией, не было. Вместо этого был создан `Animator Controller`, в котором указано, что анимация запускается, как только объект оказывается в сцене и повторяется постоянно.

Чтобы будка летела в космосе, очевидно, нужен космос. В Assets Store был найден подходящий `skybox` космоса, который и был использован.

Но ощущения полета по-прежнему не было. Всё потому что сам `skybox` оставался неподвижным, как и звезды на нём. Отчего будка выглядела отчаянно барахтающейся в пустоте.

Для анимации звезд была создана система частиц и написан скрипт для неё.

**Скрипт:** StarSpace, MoveAhead.

**Использовался язык программирования:** C#.

**Объект:** Скрипт StarSpace лежит на системе частиц. Скрипт MoveAhead лежит на камере и полицейской будке.

**В игре:** Во время движения камеры вокруг неё генерируются частицы, тем


Рисунок 20 – Главное меню создаваемой игры

самым обеспечивая ощущение, что будка летит мимо звезд (хотя фактически так и есть).

В итоге главное меню игры включает в себя анимированный фон, название и кнопки навигации. На скриншотах можно заметить, что будка и звезды действительно двигаются (Рисунок 20).

## 5.5 Создание игрового меню

По тем же причинам, что и главное меню, было принято решение, и игровое меню делать в футуристичной стилистике, минимализме и оттенках синего. Шрифт был выбран тот же, что и в главном меню, чтобы сохранить стилистику.

### 5.5.1 Игровое меню

**Скрипт:** GamingMenu.

**Использовался язык программирования:** C#.

**Объект:** Скрипт GaimingMenu лежит на игровом персонаже.

**В игре:** Когда игрок нажимает на кнопку escape, игровой мир встает на паузу, на экран накладывается полупрозрачная синяя текстура и в центре экрана (не зависимо от размеров и разрешения окна) появляются кнопки навигации. Для сравнения приводится скриншот, где, для наглядности, правая половина экрана затемнена, а левая нет (Рисунок 21).

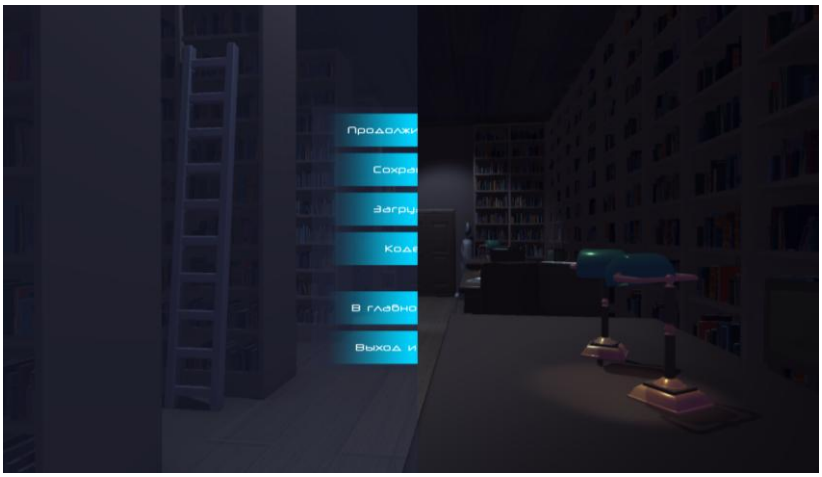

Рисунок 21 – Экран в режиме паузы (левая половина)
и в режиме игры (правая половина)

При повторном нажатии на escape или нажатии на кнопку «Продолжить игру», игровой мир снимается с паузы, а меню снова становится невидимым. При нажатии на кнопку «Выход в главное меню» происходит переход к сцене главного меню. При нажатии на кнопку «Выход» происходит выход из игры. В обоих случаях игрок предупреждается, что несохраненные изменения будут утрачены.

При нажатии на кнопку «Сохранение» открывается меню сохранения. При нажатии на кнопку «Загрузка» открывается меню загрузки. При нажатии на кнопку «Кодекс» открывается кодекс текущей игры.

### 5.5.2 Сохранение и загрузка

**Скрипт:** GamingMenu, testSaveLoad.

**Использовался язык программирования:** C#.

**Объект:** Скрипты GamingMenu и testSaveLoad лежат на игровом персонаже.

**В игре:** При нажатии кнопки «Сохранение» или «Загрузка» права передаются скрипту testSaveLoad с соответствующим маркером, указывающим, является ли это вызовом меню сохранения или загрузки. В любом случае на экране отображается список доступных сохранений и кнопки «назад» и «Сохранить»/«Загрузить» в зависимости от того, какое


Рисунок 22 – Меню загрузки

именно меню отображается.

При сохранении выбирается, какое сохранение надо переписать, после чего кнопка «Сохранить» вызывает окно подтверждения.

Данные о текущей игре записываются в указанный слот и сохраняются в файл игровых сохранений.

При загрузке выбирается, какое сохранение необходимо загрузить, после чего кнопка «Загрузить» вызывает окно подтверждения. Выбранное сохранение замещает текущее состояние игрового мира, загружаются соответствующие уровень и координаты персонажа.

### 5.5.3 Кодекс

Кодекс представляет собой свод заметок и информации, которую игрок нашел за время своего прохождения. Не важно, получил ли он её в ходе диалогов или изучения объектов окружения.

**Скрипт:** GamingMenu, XMLtest, testSaveLoad.

**Использовался язык программирования:** C#.

**Объект:** Скрипты GamingMenu, XMLtest и testSaveLoad лежат на игровом персонаже.

**В игре:** При нажатии на кнопку «Кодекс» в игровом меню скрипт GamingMenu передает управление скрипту XMLtest. Программа получает

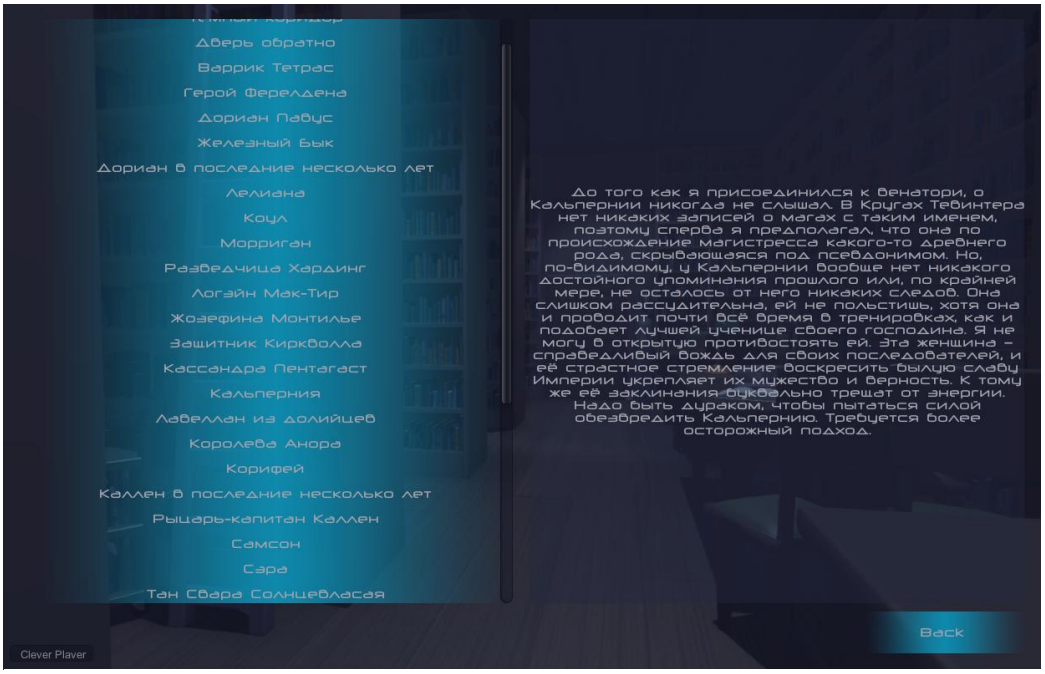


Рисунок 23 – Кодекс с большим количеством записей

информацию о найденных заметках кодекса из скрипта testSaveLoad. Она сопоставляет их общему списку заметок кодекса (хранящемуся в формате xml). В левой части экрана отображаются только те заметки, которые были найдены в ходе прохождения текущего сохранения. При нажатии на заметку кодекса в списке слева, в правой части экрана отображается полный текст заметки. Если список найденных заметок слишком велик и не помещается на экран, то появляется возможность прокрутки списка (рис. 23- 24).


Рисунок 24 – Кодекс с малым количеством записей

В правом нижнем углу находится кнопка «Назад», которая возвращает к игровому меню. При нажатии на кнопку «Escape» игровой мир снимается с паузы, а меню кодекса перестает отображаться.

### 5.6 Диалоговая система

Диалоги являются важной частью любой игры выбранного жанра. Особенно при учете, что был выбран именно формат разветвленного сюжета и системы взаимоотношений с персонажами. Во время диалогов можно повысить или, наоборот, понизить уровень доверия с теми или иными персонажами, выбирая соответствующие реплики. Некоторые диалоги могут быть прослушаны более одного раза (приветствие), некоторые лишь одиножды (дающие развитие сюжету). Кроме того важно, чтобы NPC реагировал

корректно в зависимости от того, какая реплика была выбрана героем. Важно чтобы выбор реплики игроком действительно на что-то влиял, хоть и в таких мелочах. Всё это необходимо было учесть при создании скриптов.

Решено было, что ответы NPC на реплики игрока будут в основном строиться по принципу: реакция на конкретную реплику игрока, после неё общее для всех ответов продолжение разговора.

**Скрипт:** GamingMenu, DialogText, testSaveLoad.

**Использовался язык программирования:** C#.

**Объект:** Скрипты GamingMenu, DialogText и testSaveLoad лежат на игровом персонаже.

**В игре:** Когда персонаж оказывается лицом к герою, с которым возможен диалог, и при этом он достаточно близко для этого, внизу экрана отображается подсказка о том, что можно начать диалог. При активации диалога, запускается анимация разговора, персонаж больше недоступен для перемещений, мышка становится активна. В верхней части экрана отображаются реплики NPC, в нижней части экрана представлены реплики,

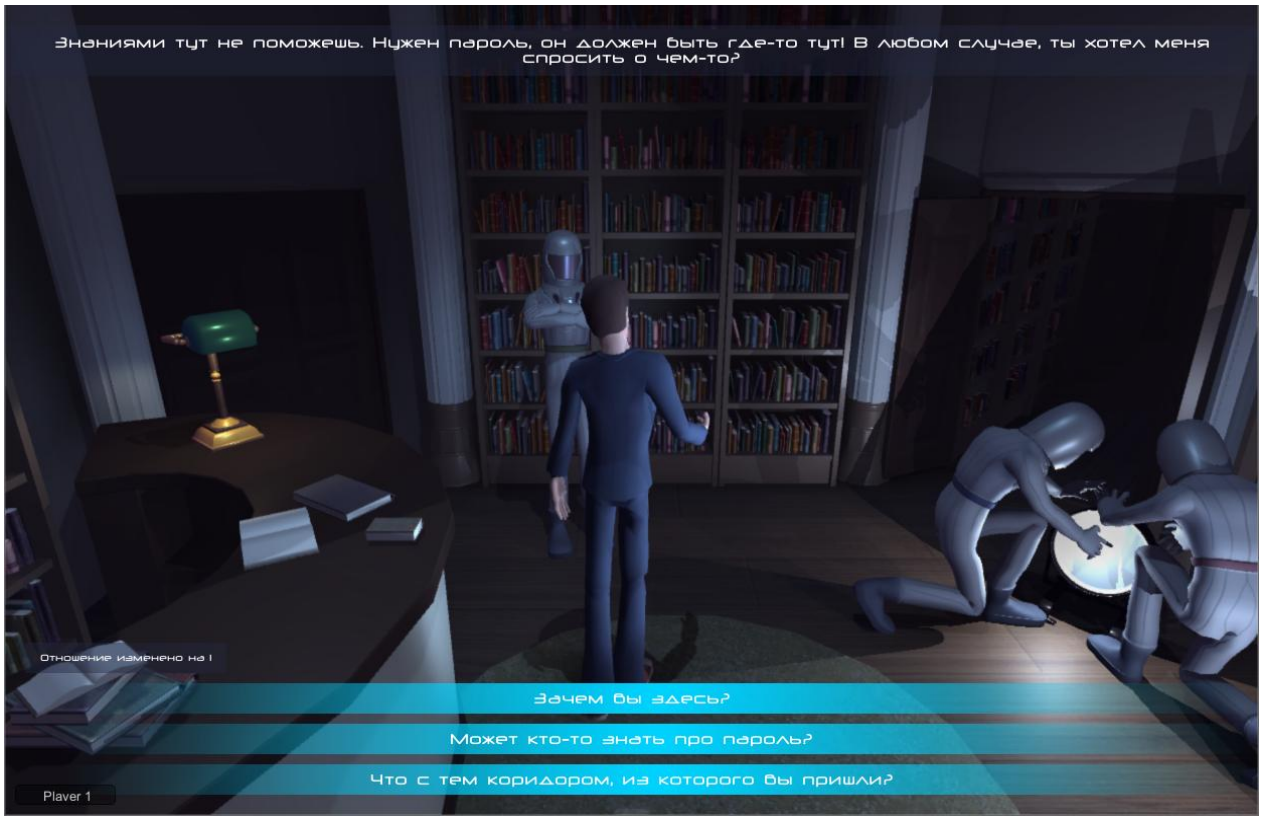


Рисунок 25 – Пример диалога

доступные для главного героя.

Диалоги каждого персонажа для каждой отдельной сцены хранятся в отдельном xml-документе с соответствующей внутренней структурой. Некоторые диалоговые реплики доступны только тогда, когда уровень доверия конкретного персонажа необходимого уровня или выше. Так же как и некоторые ветки диалога (реплики самого NPC). Некоторые из реплик позволяют изменить уровень доверия с персонажем. Если это происходит, то слева появляется сообщение о том, что отношение изменено на определенное количество очков (в плюс или в минус).

## Заключение

В ходе данной выпускной квалификационной работы была создана первая глава игры «Доктор Кто: Тишина в библиотеке».

После анализа и рассмотрения была выбрана среда разработки Unity3D, как самая подходящая для условий этого проекта.

Подходящим, актуальным и удовлетворяющим потенциальных игроков был выбран жанр «Action-adventure с разветвлённым сюжетом», то есть включающий несколько концовок, зависящих от решений игрока.

Ну а основной игровой платформой для игры был выбран ПК, но с возможностью играть и с использованием альтернативных пользовательских интерфейсов.

Кроме того, был принят ряд решений и относительно создания массовых моделей среды (на примере построения библиотеки). Часть массовых моделей будет дублироваться и расставляться вручную, а часть генерироваться программным образом.

Все решения были приняты обоснованно, подтверждаемые мнением потенциальной аудитории, опросом и примерами из популярных, существующих компьютерных игр.

После чего, используя средства Unity, удалось импортировать все необходимые модели и собрать из них полноценные игровые сцены, включающие объекты для взаимодействия разных типов, освещение и неактивные объекты, необходимые для оформления и создания интерьера.

Кроме того, камера игры была настроена так, чтобы комнаты выглядели правдоподобнее и приятнее для игрока.

Также была осуществлена настройка управления персонажем через клавиатуру и контроллер (протестировано при помощи контроллера Xbox One). И создана возможность взаимодействия персонажа с объектами двух типов. Помимо этого была подключена возможность внутриигрового отображения видео на экране телевизора.

В ходе работы было также создано главное меню. В качестве фона для меню была использована модель с зацикленной анимацией и система частиц, настроенная при помощи скрипта. Кроме того были реализованы алгоритмы автоматической генерации книг на полках. Была обеспечена работа кодекса и функционирование возможностей сохранения и загрузки.

## Список использованных источников


1. Unity – Game Engine [Электронный ресурс] / Режим доступа: https://unity3d.com/ru, свободный. – Загл. с экрана.
2. Разработка|App2Top / Режим доступа: http://www.unity3d.ru свободный. – Загл. с экрана.
3. Обзор самых популярных движков для разработки игр – «Хакер» [Электронный ресурс] / Режим доступа: https://xakep.ru/2014/09/05/game-development-engines-review/, свободный. – Загл. с экрана.
4. What is Unreal [Электронный ресурс] / Режим доступа: https://www.unrealengine.com/, свободный. – Загл. с экрана.
5. CryEngine - Википедия [Электронный ресурс] / Режим доступа: https://ru.wikipedia.org/wiki/CryEngine, свободный. – Загл. с экрана.
6. CryEngine|Crytek [Электронный ресурс] / Режим доступа: http://www.crytek.com/cryengine/cryengine3/overview, свободный. – Загл. с экрана.
7. Разработка|App2Top / Режим доступа: http://app2top.ru/game_development, свободный. – Загл. с экрана.
8. Unity – Руководство: Как мне импортировать модели из моего 3D редактора? [Электронный ресурс] / Режим доступа: http://docs.unity3d.com/ru/current/Manual/HOWTO-importObject.html, свободный. – Загл. с экрана.
9. Занимательная физика – Статьи – Unity 3D [Электронный ресурс] / Режим доступа: http://xgm.guru/p/unity/unityphysics, свободный. – Загл. с экрана.

10. 10 интересных эффектов [Электронный ресурс] / Режим доступа: http://forum.ggeek.ru/index.php?/topic/1956-10-интересных-эффектов/, свободный. – Загл. с экрана.
11. Xbox 360 Controller – Unify Community Wiki [Электронный ресурс] / Режим доступа: http://wiki.unity3d.com/index.php?title=Xbox360Controller, свободный. – Загл. с экрана.
12. Unity model import – Terry Morgan [Электронный ресурс] / Режим доступа https://sites.google.com/site/terrymorgan1213/tutorials/unity_model_import, свободный. – Загл. с экрана.

# Приложение

## Приложение 1. Скрипт gamingMenu

Скрипт, отвечающий за корректную работу игрового меню и взаимодействие персонажа с игровым миром. Язык программирования: C#

```
using UnityEngine;
using System.Collections;
public class gamingMenu : MonoBehaviour {
 public bool paused = false;
 public bool menuButtonsOn = true;
 private int test = 0;
 public GUIStyle customButton;
 public GUIStyle customBack;
 public GUIStyle customText;
 public GUIStyle customText2;
 private testSaveLoad tsl;
 private XMLtest xmlt;
 private DialogText dt;
 private UsableThing ut;
 private Game currentGame;
 public bool shouldICheck = true;
 private bool speakerFound = false;
 public bool usableFound = false;
 private float speakerTimer = 0.0f;
 private float speakerOffTime = 2f;
 private bool pauseEnabled = true;

 void Start () {
  tsl = GetComponent<testSaveLoad>();
  xmlt = GetComponent<XMLtest>();
  currentGame = testSaveLoad.currentGame;
 }

 void OnGUI () {
  if(pauseEnabled)
   if(Input.GetButtonUp("Escape")) {
    test ++;
    if (test % 2 == 0){
     if(!paused) pauseOn();
     else pauseOff();
    }
   }
```

```
  if(paused) {
   GUI.Box(new Rect(0,0,Screen.width,Screen.height),
"", customBack);
   if(menuButtonsOn) {
    GUI.Box(new Rect(Screen.width/2-
150,Screen.height/2-200, 300, 400), "", customBack);
    if(GUI.Button(new Rect(Screen.width/2-
140,Screen.height/2-190,280,50), "Продолжить игру",
customButton)) pauseOff();
    if(GUI.Button(new Rect(Screen.width/2-
140,Screen.height/2-130,280,50), "Сохранить",
customButton)) saveBtn();
    if(GUI.Button(new Rect(Screen.width/2-
140,Screen.height/2-70,280,50), "Загрузить",
customButton)) loadBtn();
    if(GUI.Button(new Rect(Screen.width/2-
140,Screen.height/2-10,280,50), "Кодекс",
customButton)) codexOn();
    if(GUI.Button(new Rect(Screen.width/2-
140,Screen.height/2+80,280,50), "В главное меню",
customButton)) mainMenu();
    if(GUI.Button(new Rect(Screen.width/2-
140,Screen.height/2+140,280,50), "Выход из игры",
customButton)) gameExit();
   }
  } else if (speakerFound) {
   GUI.Box(new Rect(0,Screen.height-
100,Screen.width,70), "Нажмите E, чтобы поговорить",
customBack);
   if(Input.GetButtonUp("Action")) {dialogOn();}
  } else if(usableFound) {
   GUI.Box(new Rect(0,Screen.height-
100,Screen.width,70), ut.tipStr, customBack);
   if(Input.GetButtonUp("Action")) {newCodexOn();}
  }
 }

 public void newCodexOn() {
  ut.work = true;
  pauseEnabled = false;
  usableFound = false;
  Cursor.visible = true;
this.GetComponent<UnityStandardAssets.Characters.FirstP
erson.FirstPersonController>().enabled = false;
  Time.timeScale = 0;
```

```
  speakerFound = false;
  usableFound = false;
  shouldICheck = false;
 }

 public void newCodexOff() {
  pauseEnabled = true;
  Cursor.visible = false;
this.GetComponent<UnityStandardAssets.Characters.FirstP
erson.FirstPersonController>().enabled = true;
  Time.timeScale = 1;
  shouldICheck = true;
 }

 public void dialogOn() {
  pauseEnabled = false;
  dt.startDialog();
this.GetComponent<UnityStandardAssets.Characters.FirstP
erson.FirstPersonController>().enabled = false;
  Cursor.visible = true;
  speakerFound = false;
  shouldICheck = false;
 }

 public void dialogOff() {
  pauseEnabled = true;
this.GetComponent<UnityStandardAssets.Characters.FirstP
erson.FirstPersonController>().enabled = true;
  Cursor.visible = false;
  shouldICheck = true;
 }

 void Update() {
  if(shouldICheck) {
   RaycastHit hit;
   Vector3 fwd =
transform.TransformDirection(Vector3.forward);
   if (Physics.Raycast(transform.position, fwd, out
hit, 4)) {
    if (hit.collider.tag == "Speaker") {
     speakerFound = true;
     usableFound = false;
     dt = hit.collider.GetComponent<DialogText>();
    } else if (hit.collider.tag == "Tv") {
     ut = hit.collider.GetComponent<UsableThing>();
```

```
      usableFound = true;
      speakerFound = false;
     } else if (hit.collider.tag == "InfoThings") {
      ut = hit.collider.GetComponent<UsableThing>();
if(!testSaveLoad.currentGame.knowledgeArray[ut.id]) {
       usableFound = true;
       speakerFound = false;
      }
     }
    } else {
     speakerFound = false;
     usableFound = false;
    }
   }
  }

  private void codexOn() {
   menuButtonsOn = false;
   xmlt.isPressed = true;
   xmlt.isChosenKnowledgeElement = false;
  }

  private void saveBtn() {
   menuButtonsOn = false;
   tsl.activateMenu = true;
   tsl.isItForSave = true;
  }

  private void loadBtn() {
   menuButtonsOn = false;
   tsl.activateMenu = true;
   tsl.isItForSave = false;
  }

  private void pauseOn() {
   paused = true;
   Cursor.visible = true;
this.GetComponent<UnityStandardAssets.Characters.FirstP
erson.FirstPersonController>().enabled = false;
   Time.timeScale = 0;
   speakerFound = false;
   usableFound = false;
   shouldICheck = false;
  }
```

```
 public void pauseOff() {
  paused = false;
  xmlt.isPressed = false;
  Cursor.visible = false;
this.GetComponent<UnityStandardAssets.Characters.FirstP
erson.FirstPersonController>().enabled = true;
  Time.timeScale = 1;
  shouldICheck = true;
 }

 private void mainMenu() {
  Cursor.visible = true;
  Application.LoadLevel(0);
 }

 private void gameExit() {
  Application.Quit();
  }
}
```

**Приложение 2. Скрипт XMLtest**

Скрипт, работающий с кодексом, который хранится в xml-документе. Язык программирования: C#.

```
using UnityEngine;
using System;
using System.Collections;
using System.Xml;
using System.IO;
using System.Collections.Generic;
using System.Text.RegularExpressions;

public class XMLtest : MonoBehaviour {
 public TextAsset asset;
 public string text1;
 public string text2;
 private XmlTextReader reader;
 public static List<KnowledgeElement> knowledgeLibrary
= new List<KnowledgeElement>();
 private bool isKnow = false;
 public Vector2 scrollview =Vector2.zero;
 private int currentNum = -1;
 public bool isPressed = false;
 public bool isChosenKnowledgeElement = false;
 public static int idk;
 int j;
 private gamingMenu gm;
 private GUIStyle customButton;
 private GUIStyle customBack;
 private GUIStyle customText;

 void Start () {
  parseXML();
  idk = knowledgeLibrary.Count;
  gm = GetComponent<gamingMenu>();
  customButton = gm.customButton;
  customBack = gm.customBack;
  customText = gm.customText;
 }

 void OnGUI() {
  if(isPressed) {
   if(Input.GetButtonUp("Escape")) {backBtn();}
```

```
   if(GUI.Button(new Rect(Screen.width - 220,
Screen.height - 70, 200, 50), "Back", customButton))
    backBtn();
   if(!isChosenKnowledgeElement)
    GUI.Box (new Rect(Screen.width/2 +
10,20,Screen.width/2 - 30,Screen.height - 100), "
",customText);
   else
    GUI.Box (new Rect(Screen.width/2 +
10,20,Screen.width/2 - 30,Screen.height - 100),
knowledgeLibrary[currentNum].mainText,customText);
    j = 0;
    scrollview = GUI.BeginScrollView(new
Rect(20,20,Screen.width/2 - 30,Screen.height - 100),
scrollview, new Rect(0, 0, 400,
findHeightOfScrollView()));
    for(int i = 0; i < knowledgeLibrary.Count; i++) {
     if ((knowledgeLibrary[i] !=
null)&&(testSaveLoad.currentGame.knowledgeArray[i])) {
      if(GUI.Button(new Rect(0,j,Screen.width/2 -
30,30), knowledgeLibrary[i].titleName, customButton)) {
       currentNum = i;
       isChosenKnowledgeElement = true;
      }
      j += 30;
     }
    }
    GUI.EndScrollView();
   }
  }

 private void backBtn() {
  gm.menuButtonsOn = true;
  isPressed = false;
 }

 private void parseXML() {
  reader = new XmlTextReader(new
StringReader(asset.text));
  while(reader.Read()) {
   if(reader.Name == "text") {
    text1 = reader.GetAttribute("label1");
    text2 = reader.GetAttribute("label2");
   }
   if(reader.Name == "knowledge") isKnow = !isKnow;
```

```
   if((reader.Name == "item") && isKnow) {
    KnowledgeElement tmp = new KnowledgeElement();
    tmp.titleName = reader.GetAttribute("titleName");
    tmp.mainText = reader.GetAttribute("mainText");
    string[] stringSeparators = new string[] {"/n"};
    string[] result;
    result = tmp.mainText.Split(stringSeparators,
StringSplitOptions.None);
    tmp.mainText = String.Join("\n", result, 0,
result.Length);
    knowledgeLibrary.Add(tmp);
   }
  }
 }

 private int findHeightOfScrollView() {
  int k = 30;
   foreach(bool each in
testSaveLoad.currentGame.knowledgeArray)
    if(each)
     k += 30;
    return k;
 }
}
```

**Приложение 3. Скрипт UsableThing**

Срипт, отвечающий за взаимодействие персонажа с объектами. Язык программирования: C#.

```
using UnityEngine;
using System.Collections;

public class UsableThing : MonoBehaviour {
 private bool tipIsOn = false;
 private float tipTimer = 0.0f;
 private float tipOnTime = 3f;
 public string tipStr;
 public string tipStr2;
 public MovieTexture movie;
 public Texture offTexture;
 public Material myMaterial;
 public int id;
 private gamingMenu gm;
 private GUIStyle customButton;
 private GUIStyle customBack;
 private GUIStyle customText;
 private GUIStyle customText2;
 public bool work = false;
 private bool tvIsOff = true;

 void Start () {
  gm =
GameObject.Find("Guy").GetComponent<gamingMenu>();
  customButton = gm.customButton;
  customBack = gm.customBack;
  customText = gm.customText;
  customText2 = gm.customText2;
  if(this.tag == "tv") {
   movie.Stop();
   myMaterial.mainTexture = offTexture;
  }
 }

 void OnGUI () {
  if(work) {
   if(this.tag == "Tv") {
    work = false;
    gm.newCodexOff();
    string tmp = tipStr;
```

```
    tipStr = tipStr2;
    tipStr2 = tmp;
   if(tvIsOff) {
    myMaterial.mainTexture = movie;
    movie.Play();
    tvIsOff = false;
   } else {
    myMaterial.mainTexture = offTexture;
    movie.Stop();
    tvIsOff = true;
   }
  } else if(this.tag == "InfoThings") {
   testSaveLoad.currentGame.knowledgeArray[id] = true;
   GUI.Box(new Rect(Screen.width/2-250,Screen.height/2-
200,500,400), "", customBack);
   GUI.Box(new Rect(Screen.width/2-250,Screen.height/2-
200,500,30), "", customBack);
   GUI.Box(new Rect(Screen.width/2-250,Screen.height/2-
200,500,30), XMLtest.knowledgeLibrary[id].titleName,
customBack);
   GUI.Box(new Rect(Screen.width/2-250,Screen.height/2-
170,500,340), XMLtest.knowledgeLibrary[id].mainText,
customText);
   if((GUI.Button(new Rect(Screen.width/2-
250,Screen.height/2+170,500,30), "Ok", customButton))
|| (Input.GetButtonUp("Escape"))) {
    work = false;
    tipIsOn = true;
    gm.newCodexOff();
   }
  }
 }
 if(tipIsOn) {
  if(Input.GetButtonUp("Escape")) {tipIsOn = false;}
  tipTimer += Time.deltaTime;
  GUI.Box(new Rect(Screen.width-330,Screen.height-
40,300,30), "Запись была добавлена в кодекс",
customText2);
  }
  if(tipTimer > tipOnTime) {
   tipIsOn = false;
  }
 }
}
```

**Приложение 4. Скрипт testSaveLoad**

Скрипт, отвечающий за функции сохранения и загрузки. Язык программирования: C#.

```
using System;
using UnityEngine;
using UnityEngine.UI;
using System.Collections;
using System.Collections.Generic;
using System.Runtime.Serialization.Formatters.Binary;
using System.IO;

public class testSaveLoad : MonoBehaviour {
 public static List<Game> savedGames = new
List<Game>();
 private string saveDataPath;
 private string saveDirPath;
 public static Game currentGame;
 private gamingMenu gm;
 public bool activateMenu = false;
 public Vector2 scrollview =Vector2.zero;
 public string stringToEdit = " ";
 private bool setActiveInputMenu = false;
 public bool isItForSave = false;
 private int currentNum;
 private bool saveMenuActive = false;
 private bool loadMenuActive = false;
 private bool isEmpty = true;

 void Start () {
  gm = GetComponent<gamingMenu>();
  saveDirPath = Application.streamingAssetsPath + "/New
Saves";
  saveDataPath = saveDirPath + "/savedGames.dw";
  loadGames();
  if(!isEmpty)
   currentGame =
savedGames[PlayerPrefs.GetInt("CurrentSave")];
  else
   currentGame = new Game();
 }

 void OnGUI() {
```

```
  GUI.Box(new Rect(10,Screen.height-30,100,20),
currentGame.name);
  if(activateMenu){
   if(!isEmpty)
    GUI.Box(new Rect(Screen.width/2-
400,Screen.height/2-280 , 800, 400),
savedGames[currentNum].name, gm.customBack);
   else
    GUI.Box(new Rect(Screen.width/2-
400,Screen.height/2-280 , 800, 400), "NEW GAME",
gm.customBack);
    GUI.Box (new Rect(Screen.width/2-
200,Screen.height/2+120,400,160), " ");
    int j = 0;
    scrollview = GUI.BeginScrollView(new
Rect(Screen.width/2-200,Screen.height/2+120,420,160),
scrollview, new Rect(0, 0, 400,
findHeightOfScrollView()));
    if(!isEmpty)
     for(int i = savedGames.Count - 1; i > -1; i--, j
+= 30) {
      if (savedGames[i] != null) {
       if(GUI.Button(new Rect(0,j,400,30),
savedGames[i].name + " - " +
savedGames[i].lastGameDate, gm.customButton))
        currentNum = i;
       }
      }
      GUI.EndScrollView();
      if(GUI.Button(new Rect(Screen.width - 120, 20,
100, 20), "back",gm.customButton))
       backBtn();
       if(isItForSave) {
        if(GUI.Button(new Rect(Screen.width - 120,
Screen.height - 45, 100, 20), "SAVE",gm.customButton))
         saveMenuActive = true;
        } else {
         if(GUI.Button(new Rect(Screen.width - 120,
Screen.height - 45, 100, 20), "LOAD",gm.customButton))
          loadMenuActive = true;
         }
        }
        if(saveMenuActive) {
         GUI.Box (new Rect(Screen.width/2-
150,Screen.height/2-100,300,140), "Save game in \"" +
```

```
savedGames[currentNum].name + "\" slot?",
gm.customBack);
         if(GUI.Button(new Rect(Screen.width/2-
140,Screen.height/2 - 15,50,50), "Ok",gm.customButton))
{
          currentGame.lastGameDate = DateTime.Now;
          savedGames[currentNum] = currentGame;
          CloseMenu();
          saveMenuActive = false;
          backBtn();
         }
     if(GUI.Button(new
Rect(Screen.width/2+50,Screen.height/2 - 15,60,50),
"Cancel",gm.customButton)) {saveMenuActive = false;}
    }
    if(loadMenuActive) {
     GUI.Box (new Rect(Screen.width/2-
150,Screen.height/2-100,300,140), "Load game \"" +
savedGames[currentNum].name +"\"?", gm.customBack);
     if(GUI.Button(new Rect(Screen.width/2-
140,Screen.height/2 - 15,50,50), "Ok",gm.customButton))
{
      PlayerPrefs.SetInt("CurrentSave", currentNum);
      loadMenuActive = false;
      currentGame =
savedGames[PlayerPrefs.GetInt("CurrentSave")];
      backBtn();
     }
     if(GUI.Button(new
Rect(Screen.width/2+50,Screen.height/2 - 15,60,50),
"Cancel",gm.customButton)) {
      loadMenuActive = false;
     }
    }
    if(setActiveInputMenu) {
     activateMenu = false;
     GUI.Box (new Rect(Screen.width/2-
150,Screen.height/2-100,300,140), "Input Name",
gm.customBack);
     stringToEdit = GUI.TextField(new
Rect(Screen.width/2-100, Screen.height/2-70, 200, 20),
stringToEdit, 25);
     if(GUI.Button(new Rect(Screen.width/2-
140,Screen.height/2 - 15,50,50),
"Ok",gm.customButton)){
```

```
      setActiveInputMenu = false;
      currentGame.name = stringToEdit;
      currentGame.lastGameDate = DateTime.Now;
      savedGames.Add(currentGame);
      CloseMenu();
     }
     if(GUI.Button(new
Rect(Screen.width/2+50,Screen.height/2 - 15,60,50),
"Cancel",gm.customButton)){
      setActiveInputMenu = false;
      activateMenu = true;
     }
  }
 }

 private void loadGames() {
  if(File.Exists(saveDataPath)) {
   isEmpty = false;
   BinaryFormatter bf = new BinaryFormatter();
   FileStream file = File.Open(saveDataPath,
FileMode.Open);
   savedGames = (List<Game>)bf.Deserialize(file);
   file.Close();
  }
 }

 public void CloseMenu() {
  BinaryFormatter bf = new BinaryFormatter();
  FileStream file;
  if (!File.Exists(saveDataPath)) {
   var folder = Directory.CreateDirectory(saveDirPath);
   file = new FileStream(saveDataPath,
FileMode.Create);
  } else {
   file = new FileStream(saveDataPath, FileMode.Open);
  }
  bf.Serialize(file, savedGames);
  file.Close();
}

 private int findHeightOfScrollView() {
  return savedGames.Count * 30;
 }
}
```

**Приложение 5. Скрипт DialogText**

Срипт отвечает за управеление диалогами для каждого конкретного персонажа. Язык программирования: C#.

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using System.Xml;
using System.IO;
using System;

public class DialogText : MonoBehaviour {
 public List<Dialog> dialoges = new List<Dialog>();
 private XmlTextReader reader;
 public TextAsset asset;
 public bool work = false;
 private string textBase;
 private int i = 0;
 private int j = 0;
 private int currentGameTL = 3;
 private float timer = 0.0f;
 private float timerMax = 3.0f;
 private bool isTimer = false;
 private int changedTL;
 private bool ifYes = false;
 private gamingMenu gm;
 private GUIStyle customButton;
 private GUIStyle customBack;
 private GUIStyle customText;
 private int w;

 void Start () {
gm = GameObject.Find("Guy").GetComponent<gamingMenu>();
  customButton = gm.customButton;
  customBack = gm.customText;
  customText = gm.customText2;
 }

 public void startDialog() {
  parseXMLDialog();
  work = true;
  textBase = dialoges[0].steps[0].textBase.textItself;
 }
```

```
 void OnGUI() {
  if(work) {
   if((dialoges[i].steps[j].textBase.minTL != 0) &&
(currentGameTL < dialoges[i].steps[j].textBase.minTL))
{
   j ++;
   } else if ((dialoges[i].steps[j].textBase.minTL !=
0)) {
    ifYes = true;
   }
   GUI.Box(new Rect(0,20,Screen.width,50), textBase,
customBack);
   if(dialoges[i].steps[j].textBase.isEnd) {
    if(GUI.Button(new Rect(0,Screen.height-
70,Screen.width,50), "Завершить разговор",
customButton)) {
     print("конец");
     textBase = "";
     j = 0;
     work = false;
     gm.dialogOff();
    }
   } else {
    w = 50;
    for(int k = 0; k <
dialoges[i].steps[j].answers.Count; k++, w += 40)
     if(currentGameTL >
dialoges[i].steps[j].answers[k].minTL)
      if(GUI.Button(new Rect(0,Screen.height -
w,Screen.width,30),
dialoges[i].steps[j].answers[k].textItself,
customButton)) {
       changedTL = dialoges[i].steps[j].answers[k].tl;
       currentGameTL += changedTL;
      textBase = dialoges[i].steps[j].replies[k] + " ";
       j ++;
       if(ifYes) {
        j++;
        ifYes = false;
       }
  textBase += dialoges[i].steps[j].textBase.textItself;
       if(changedTL != 0 ) {
        isTimer = true;
        timer = 0.0f;
       }
```

```
      }
     }
  }
  if(isTimer) {
   GUI.Box(new Rect(20,Screen.height - w,200,30),
"Отношение изменено на " + changedTL.ToString(),
customText);
   timer += Time.deltaTime;
  }
  if(timer > timerMax)
   isTimer = false;
 }

 private void parseXMLDialog() {
  string replyTMP;
  Answer answerTMP = new Answer();
  TextBase textBaseTMP = new TextBase();
  Step stepTMP = new Step();
  stepTMP.answers = new List<Answer>();
  Dialog dialogTMP = new Dialog();
  reader = new XmlTextReader(new
StringReader(asset.text));
  reader.ReadStartElement("main");
  while((reader.Read())&&(reader.Name != "main")) {
   dialogTMP = new Dialog();
   dialogTMP.steps = new List<Step>();
   reader.ReadStartElement("dialog");
   while((reader.Read()) && reader.Name != "dialog") {
    stepTMP = new Step();
    reader.ReadStartElement("step");
    while((reader.Read()) && (reader.Name != "step")) {
     if(reader.Name == "textBase") {
      textBaseTMP = new TextBase();
      textBaseTMP.minTL =
Int32.Parse(reader.GetAttribute("minTL"));
      textBaseTMP.isEnd =
Convert.ToBoolean(reader.GetAttribute("isEnd"));
      textBaseTMP.textItself =
reader.GetAttribute("textItself");
      stepTMP.textBase = textBaseTMP;
     }
     if(reader.Name == "answer") {
      answerTMP = new Answer();
      answerTMP.tl =
Int32.Parse(reader.GetAttribute("tl"));
```

```
      answerTMP.minTL =
Int32.Parse(reader.GetAttribute("minTL"));
     answerTMP.textItself =
reader.GetAttribute("textItself");
     stepTMP.answers.Add(answerTMP);
    }
    if(reader.Name == "reply") {
stepTMP.replies.Add(reader.GetAttribute("textItself"));
    }
   }
   reader.ReadEndElement(); //step
   dialogTMP.steps.Add(stepTMP);
  }
  reader.ReadEndElement(); //dialog
  dialoges.Add(dialogTMP);
 }
 reader.ReadEndElement(); //main
}
}
```

**Приложение 6. Скрипт testBooks**

Скрипт, отвечающий за генерацию книг на полках. Язык программирования: Javascript.

```
#pragma strict
public var bookPrefabs : GameObject[];
public var bookMaterial : Material;
public var textureArray : Texture[];
private var bookShelfs : GameObject[];
private var bookShelf : GameObject;
private var prevX : float;
private var shelfY : float;
private var shelfZ : float;
private var randomWidth : float;
private var randomHeight : float;
private var randomColor : int;
private var shelfEnd : float;
private var isCursed : boolean = false;
private var numOnShelf : int;

function Start () {
 bookShelfs = GameObject.FindGameObjectsWithTag
("Shelf");
 for (var i : int = 0; i < bookShelfs.length; i++){
  var j : int = 0;
  bookShelf = bookShelfs[i];
  numOnShelf = Random.Range(0,4);
  var numToBegin : int;
  if (numOnShelf < 3) numToBegin = Random.Range(0,10);
   randomWidth = Random.Range(0.0f, 0.02f);
   prevX = bookShelf.transform.position.x -
bookShelf.transform.localScale.x * 0.9  + randomWidth;
   shelfY = bookShelf.transform.position.y +
bookShelf.transform.localScale.y * 0.05;
   shelfZ = bookShelf.transform.position.z -
bookShelf.transform.localScale.z * 0.5 + 0.2;
   shelfEnd = bookShelf.transform.position.x +
bookShelf.transform.localScale.x * 0.9 ;
   var shelfBooks : GameObject = new GameObject();
   shelfBooks.transform.position.x =
bookShelf.transform.position.x;
   shelfBooks.transform.position.y = shelfY;
   shelfBooks.transform.position.z =
bookShelf.transform.position.z;
```

```
   shelfBooks.name = "BookPack " + i;
   while (prevX < shelfEnd - 0.16) {
    var widthMin : float;
    var widthMax : float;
    var heightMin : float;
    var heightMax : float;
    var zMin : float = -0.05;
    var zMax : float = 0.1;
    if ((!isCursed) && (numOnShelf > 0) && (j >=
numToBegin))
     var num : int = Random.Range(0,2);
    else num = 0;
     var newBook : GameObject =
Instantiate(bookPrefabs[num], new Vector3 (prevX,
shelfY, shelfZ + Random.Range(zMin,zMax)),
Quaternion.identity);
    if (num == 1) {
     if (Random.Range(0,2) == 0) {
      widthMin = 0.3f;
      widthMax = 0.8f;
     }else {
      widthMin = -0.3f;
      widthMax = -0.8f;
     }
      heightMax = 0.8f;
      heightMin = 0.4f;
      numOnShelf --;
      isCursed = true;
     } else {
      widthMin = 0.3f;
      widthMax = 1.6f;
      heightMax = 0.8f;
      heightMin = 0.4f;
      isCursed = false;
     }
     randomWidth = Random.Range(widthMin, widthMax);
     randomHeight = Random.Range(heightMin, heightMax);
     newBook.name = "book";
     newBook.transform.localScale = new Vector3
(randomWidth, randomHeight,
newBook.transform.localScale.z);
     if (widthMin < 0)
      newBook.transform.position.x -=
newBook.transform.localScale.x * 0.25;
```

```
      var cover : GameObject =
newBook.transform.Find("cover").gameObject;
    var rend : Renderer = cover.GetComponent(Renderer);
 randomColor = Random.Range(0,textureArray.length - 1);
 rend.material.mainTexture = textureArray[randomColor];
    if(num == 0) {
     prevX = (newBook.transform.localPosition.x +
newBook.transform.localScale.x * 0.08);
    } else {
     if (widthMin > 0)
      prevX = newBook.transform.localPosition.x +
newBook.transform.localScale.x * 0.25;
     else
      prevX = newBook.transform.localPosition.x ;
    }
    newBook.transform.parent = shelfBooks.transform;
    j ++;
   }
   if (shelfEnd - prevX > 0.08) {
    randomHeight = Random.Range(0.7f, 1.0f);
    randomWidth = (shelfEnd - prevX) / 0.08;
    var newBook2 : GameObject =
Instantiate(bookPrefabs[0], new Vector3 (prevX, shelfY,
shelfZ), Quaternion.identity);
    newBook2.name = "book";
    newBook2.transform.localScale = new Vector3
(randomWidth, randomHeight,
newBook.transform.localScale.z);
    newBook2.transform.parent = shelfBooks.transform;
   }
   shelfBooks.transform.rotation.eulerAngles.y =
bookShelf.transform.rotation.eulerAngles.y + 180;
  }
}
```

Выпускная квалификационная работа выполнена мною самостоятельно. Использованные в работе материалы из опубликованной научной, учебной литературы и Интернет имеют ссылки на них.

Отпечатано в ____ экземплярах.

Библиография ____ наименований.

Один экземпляр сдан на кафедру.

Фамилия, имя, отчество и подпись студента

Дата